CS 764: Midterm Exam, Fall 2020

Please ***print*** your name below.

Last Name:
First Name:

Instructions:

1. This is an open-book, open-notes exam. You can use any material provided in this course or on the Internet.
2. You are not allowed to discuss the exam questions and solutions with others during the exam.
3. You have **72 hours** to complete this exam. Please submit your solutions to yxy@cs.wisc.edu before **5pm, 11/6/2020 (Friday)**.
4. You can directly type your solutions in this MS word document and submit it; you can also print out the exam and submit a photocopy of it with your solutions; you can also convert the exam into a pdf file and write your solutions using iPad. Make sure your answers and precise, complete, and clear.
5. For any clarification questions, please either email the instructor: yxy@cs.wisc.edu or post your questions to piazza: https://piazza.com/class/kem5t5ku1bb3uz?cid=28

(For Instructors Use)

Q1 (30 points): Join and Parallel DB	
Q2 (30 points): Query Optimization	
Q3 (30 points): Concurrency Control	
Q4 (30 points): Recovery	
TOTAL (120 points)	

Question 1. [30 points] Join and Parallel Database

Bucky is a database administrator at UW-Madison and is trying to deploy a database for the employees and departments. The database contains the following two relations where the primary keys are the underlined attributes — Emp_ID is the primary key for relation EMP and Dept_ID is the primary key for relation DEPT. EMP.Dept_ID is a foreign key to relation DEPT. The EMP relation has size of 100 GB and DEPT relation has a size of 10 GB.

EMP: (Emp_ID, Dept_ID, name) # 100 GB
DEPT: (Dept_ID, name) # 10 GB

Bucky deploys the database on a machine with sufficient disk and 1 GB of main memory (page size = 8KB) and is implementing the GRACE hash join following [Shapiro'86] to run the following query Q1

```
Q1:  SELECT *
     FROM EMP, DEPT
     WHERE EMP.Dept_ID = DEPT.Dept_ID
```

a) **[5 points]** Does this machine has a big enough memory to run this query? Please justify your answer. (Hint: you can assume the fudge facto F = 1.4 following [Shapiro'86])

b) **[5 points]** Assuming initially both relations are stored on disk, how much data will be read from and written back to disk, respectively, in order to execute the query above (i.e., Q1) using GRACE hash join? (Note that the final results need not be written back to disk)

Due to recent hires, the EMP and DEPT relations have grown to 1000GB and 100GB, respectively. Namely, now we have:

EMP: (Emp_ID, Dept_ID, name) # 1000 GB
DEPT: (Dept_ID, name) # 100 GB

Bucky decides to migrate the single-node database to a distributed database that contains 4 servers. As a result, he has to partition each relation across the 4 servers. Bucky noticed that Q1 is the only join query that will ever be executed.

c) **[10 points]** Bucky wants each partition to have the same size for good load balancing. How should the two relations be partitioned to minimize network traffic? We can assume uniformly random data distribution for both relations. With this partitioning policy, how much network traffic is incurred in order to execute Q1? (Note that we assume the query outputs incur no network traffic)

d) **[10 points]** Some database experts suggest Bucky to hash-partition both relations based on the corresponding primary keys (i.e., Emp_ID and Dept_ID for relations EMP and DEPT, respectively). If Bucky chose this partitioning scheme, how should Q1 be executed to minimize network traffic? How much network traffic would be incurred in this case? (you can assume each server has a big enough memory for local execution)

Question 2. [30 points] Query Optimization

a) [**5 points**] Many databases consider only left-deep trees when performing query optimization. Please list at least one advantage of left-deep trees over right-deep trees and bushy trees, respectively.

Advantage of left-deep trees over right-deep trees:

Advantage of left-deep trees over bushy trees:

Consider the following database schema:

EMP: (Emp_ID, Dept_ID, name)	|EMP| = 10000 tuples
DEPT: (Dept_ID, name)	|DEPT| = 100 tuples
COURSES: (C_ID, Instr_ID, name)	|COURSES| = 30000 tuples

The underlined attributes are the primary keys. EMP.Dept_ID is a foreign key to DEPT, and COURSES.Instr_ID is a foreign key to EMP.

We will run the following query on the above schema:

```
Q2:   SELECT sum(*), EMP.Emp_ID
      WHERE EMP.Emp_ID = COURSES.Instr_ID
            and DEPT.Dept_ID = EMP.Dept_ID
            and DEPT.name = "Computer Sciences"
      GROUP BY EMP.Emp_ID
```

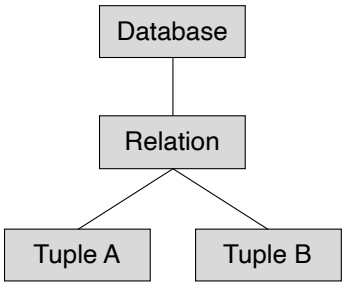
b) [**5 points**] What attributes in the relations contain "interesting orders" (defined in Lecture 4) for query Q2?

c) [**10 points**] Please draw all possible query plans of the query above (i.e., Q2). To limit the search space, we will make the following assumptions: (1) consider only nested loop joins, (2) the inner relation is always the smaller relation (Note that the right child is the inner relation by convention), (3) push down group-by as much as possible, (4) all tables are accessed using sequential scan. Your query plan should contain the following operators: table scan, join, and group-by. (Hint: the number of possible query plans should be no more than 4)

d) [**10 points**] Out of the query plans that you have generated above, which one do you think should achieve the best performance? Please justify your answer. For the execution cost, please consider only the number of comparisons in a join, i.e., joining a relation with N tuples and a relation with M tuples leads to N*M comparisons. You can assume only one department has name="Computer Sciences" and uniform data distribution (e.g., each department has the same number of employees).

Question 3. [30 points] Concurrency Control

a) [**10 points**] Consider the following locking hierarchy where there is a single database that contains a single table and the table contains two tuples: A and B. If a transaction T1 reads tuple A and writes tuple B, what lock modes (e.g., NL, S, X, IS, IX, SIX) will T1 hold on the tuples, the table, and the database, respectively?



b) [**10 points**] Now we add a transaction T2 that reads tuple B and writes tuple A. Namely:

T1: read(A) write(B)
T2: read(B) write(A)

Please write down a schedule of the two transactions that can cause a deadlock in a Strict Two-Phase Locking protocol. You can assume that a transaction always waits when a lock conflict occurs. Hint: the schedule should contain four operations: T1.lock-S(A), T1.lock-X(B), T2.lock-S(B), T2.lock-X(A) and that T1.lock-S(A) must occur before T1.lock-X(B), and T2.lock-S(B) must occur before T2.lock-X(A).

c) [**10 points**] For the schedule that you came up with in the above question, will the deadlock still occur if the two transactions are executed under **Read Committed** (RC) rather than Serializability (Note that strict 2PL implies serializability)? Please justify your answer.

Question 4. [30 points] Recovery

a) [**10 points**] Bucky is maintaining a disk-based database that implements ARIES as the recovery protocol. After one crash occurs, the database log contains the following entries since the last checkpoint. Please fill in the contents of the Transaction Table and the Dirty Page Table after the analysis phase of the recovery process. (Hint: not all rows need to be filled in both tables)

LSN	Log entries
1	Begin checkpoint (empty Transaction Table and Dirty Page Table)
2	End checkpoint
3	[Update] T1 writes P1
4	[Update] T2 writes P1
5	[Update] T2 writes P3
6	T2 commit
7	T2 end
8	T1 abort
9	CLR: UNDO T1
10	T1 end
11	Crash

Transaction Table

TransID	LastLSN

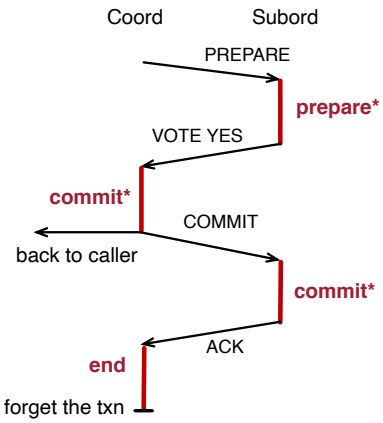
Dirty Page Table

PageID	RecLSN

b) [**10 points**] In order to achieve higher performance, Bucky decided to migrate the database to a server with a larger memory. The current database assumes **No Force** and **Steal** policies following ARIES; the new database, however, assumes **No Force** and **No Steal**. Namely, a transaction's modifications can be written back to the database on disk only when the transaction commits. Bucky realized that the original ARIES protocol can be simplified in this new design. Please write down at least two simplifications that can be made to the original ARIES protocol.

c) [**10 points**] In the following two diagrams, the first one is the original two-phase commit (2PC) protocol assuming a successful commit; the second one is a modified 2PC protocol where the coordinator sends the transaction output back to the caller before logging the local commit message. Is the modified protocol still correct? Please justify your answer. (Hint: consider potential failures)

Original 2PC



Modified 2PC